Popular Computing

Volume 8 Number 6

June 1980

87

Creative Publications, Box 10328, Palo Alto, CA 94303, markets a wide variety of materials for use in a mathematics classroom. Their catalog (112 pages, printed with excellent color photographs) lists hundreds of fascinating things, among which are these, of interest to computing teachers:

1. A poster (22 x 35 inches) displaying explicitly the 27th Mersenne prime,

$$2^{44497} - 1$$

with all its 13395 digits, together with some notes about its calculation. These sell in sets of 2 at $3.95, plus 50¢ shipping charges.

2. A poster (22 x 35 inches) showing the first 8183 significant digits of pi, together with facts on pi and the plotting of the poster by computer. $2.00 + 50¢.

3. Many hard-to-get books, including Polya's How To Solve It; books of prints by M. C. Escher; Kraitchik's Mathematical Recreations; the puzzle books of Sam Loyd; several of Martin Gardner's books; Philip Davis' Lore of Large Numbers; Bell's Men of Mathematics; the mathematics history books of D. E. Smith and W. W. Rouse Ball; and James and James' Mathematical Dictionary.

4. Plastic one-centimeter cubes (each weighing one gram) in ten colors.

5. Regular dice, blank dice, quiet dice, tiny dice, number dice, and polyhedra dice.

Publisher: **Audrey Gruenberger**
Editor: **Fred Gruenberger**
Associate Editors: **David Babcock**
Irwin Greenwald
Patrick Hall
Contributing Editors: **Richard Andree**
William C. McGee
Thomas R. Parkin
Edward Ryan
Art Director: **John G. Scott**
Business Manager: **Ben Moore**

POPULAR COMPUTING is published monthly at Box 272, Calabasas, California 91302. Subscription rate in the United States is $20.50 per year, or $17.50 if remittance accompanies the order. For Canada and Mexico, add $1.50 per year. For all other countries, add $3.50 per year. Back issues $2.50 each. Copyright 1980 by POPULAR COMPUTING.

Timer

by DAVID BABCOCK

A common problem in computing is measuring how long a program (or piece of a program) takes to run. Many computer systems need this information for accounting and billing purposes. On another level, execution time is important to know for benchmark tests, estimating production runs from test runs, measuring the effects of code changes, and so on. All large computer systems and many minicomputers typically include some form of "clock" which makes such timings easy. But most of the microcomputers now on the market don't come with such a device, at least not as standard equipment.

How can one go about timing a program accurately if his personal computer doesn't have a clock? Basically there are four alternatives. One can:

1) Buy or build a clock circuit for his computer;

2) Compute the execution time by analyzing the program and counting CPU cycles;

3) Use a stopwatch and babysit the machine while the program is running;

4) Develop a software routine which can perform the timing function without any special hardware.

The first alternative is, of course, the best solution but it takes either money (as much as $300) or a knowledge of electronics that most of us don't have. It does offer a bonus; namely, that many other uses can be made of the clock besides timing programs.

The second alternative is, in many cases, just not possible. The very nature of a problem and its solution may depend upon some very complex (or even random) interactions which are difficult, if not impossible, to predict. Most of the problems that have appeared in POPULAR COMPUTING fall into this category.

The stopwatch approach may or may not be an acceptable solution, depending on the program. For relatively quick running programs--say, under 10 minutes--it is a reasonable approach to the timing problem. However, what does one do with a program that runs 43 hours? The accuracy of your timing depends on how frequently you check on the program to see if it is still running. My experience has been that my programs finish just after I leave for the weekend.

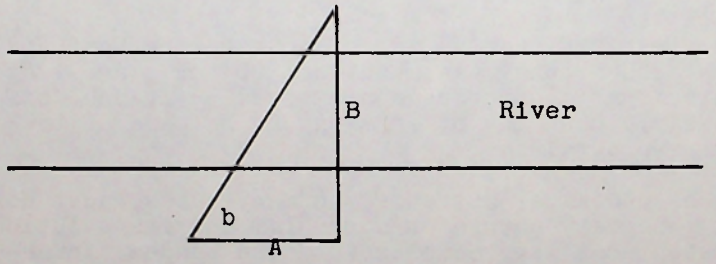
The fourth alternative--software--has some restrictions but in general is a good solution to the problem at hand. It is certainly less expensive than solution number one; it is able to time any program, especially those excluded by the second alternative; and it does not require the baby-sitting of the third alternative. Additionally, such a routine, if carefully coded, can provide a highly accurate timing base.

At first it would appear that a software solution to the timing problem is a contradiction. How can one routine time another without affecting its execution speed? More to the point, if the timer routine is running, then the program being timed is not, and vice versa.

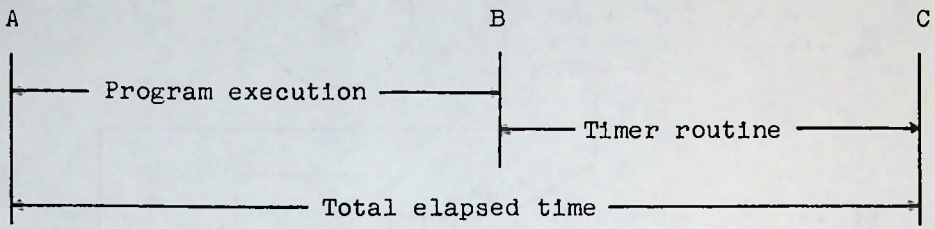
The solution to this dilemma is to draw on a "trick" used by surveyors; namely,

> If you can't directly measure what you want,
> then measure something else
> and calculate the required answer.

For example, to measure the width of a raging river, a surveyor measures off the base of a right triangle (A), measures the angle at the hypotenuse (b) and then calculates the width of the river (B) as A times tangent(b).



This same approach can be applied to the problem of timing the execution of a program. What we would like to measure is (B - A) but doing so directly is not possible. If we could measure (C - A) and (C - B), then we could deduce (B - A).



Point "A" is the time of day that the main program begins execution. The program runs uninterrupted and to completion during the time (B - A). The interval (B - C) is that period when the timer routine is running. Since the main program has ended, there is no problem with the timer routine consuming 100% of the CPU time. It just sits in a carefully measured loop, marking time. Point C is the time of day you discover that the program has ended and you terminate the timer routine.

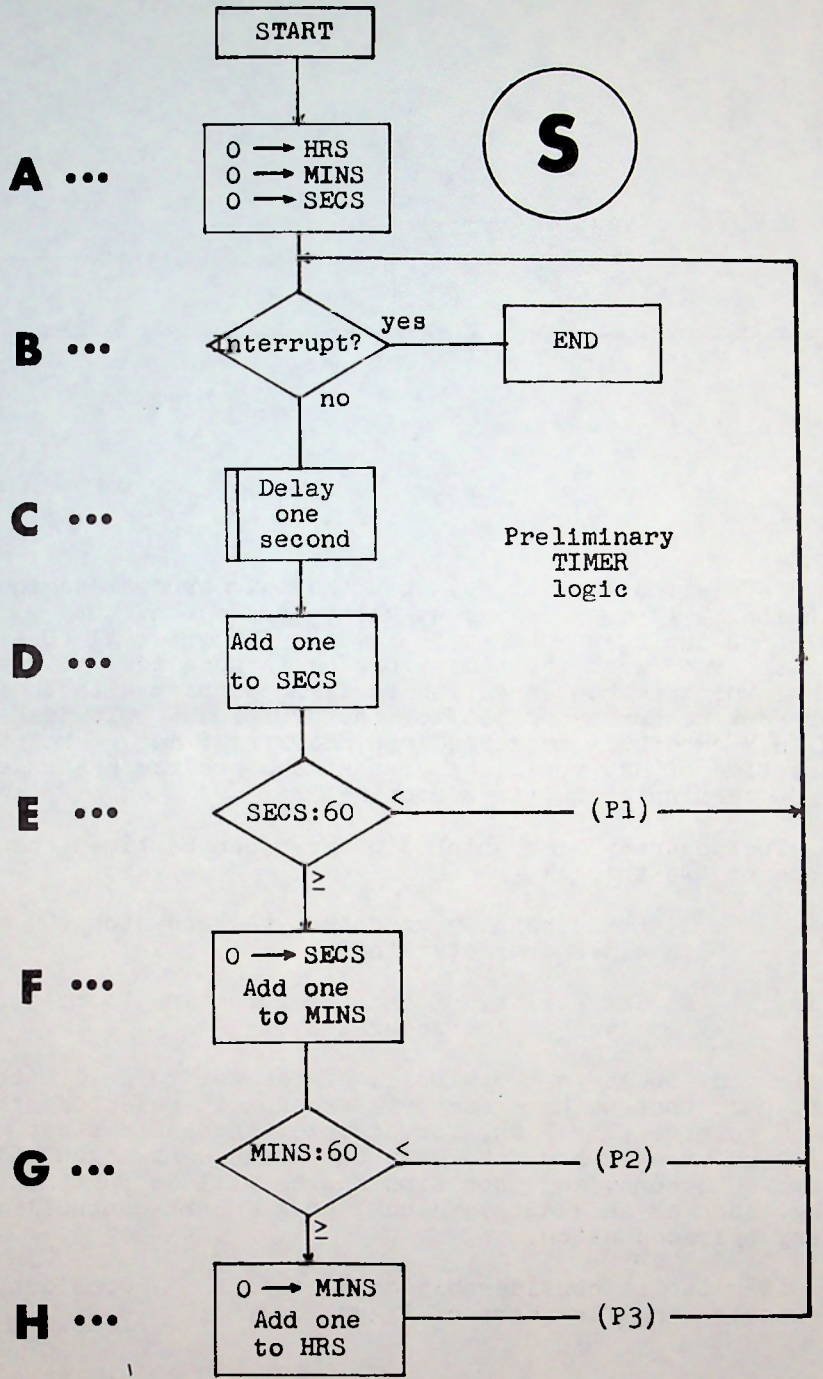
The accuracy with which a program can be timed then depends on two things:

1) How carefully the start (A) and stop (C) times are determined;

2) The ability of the timer routine to measure elapsed time accurately.

The first of these is the simpler of the two to deal with. First, note that we have complete freedom in selecting the A and C points. We can, for example, choose to start running on an even hour, to make the subsequent calculations easier. Second, any good time source will do for the timing, such as an electric clock, or a quartz-controlled battery operated watch.

Item two is considerably more complex. Consider, for example, the flowchart of Figure S.


START
S
A ...
0 → HRS
0 → MINS
0 → SECS
B ...
Interrupt?
yes
END
no
C ...
Delay one second
Preliminary TIMER logic
D ...
Add one to SECS
E ...
SECS:60
<
(P1)
≥
F ...
0 → SECS
Add one to MINS
G ...
MINS:60
<
(P2)
≥
H ...
0 → MINS
Add one to HRS
(P3)

Although it is not readily apparent, there is a basic flow in the logic of the flowchart. Notice that the box at Reference C says to delay one second. If that is done, then the inner loop (B-C-D-E-P1) will actually take longer than one second and the timer will run slow. The obvious solution is to change the delay at Reference C to be:

$$\text{one second} - B - D - E \qquad (1)$$

Note: In the discussion which follows, the letters A, B, C,... represent the execution time of the box at the corresponding reference point. For example, D represents the time it takes to execute "Add one to SECS."

The inner loop will now take precisely one second to execute. Unfortunately, we have only postponed the problem. Once per minute the additional code at References F and G will be executed. The timer is still running slow, but the error isn't as large as before. The correction is to again change the delay at Reference C, this time to:

$$\text{one second} - B - D - E - \frac{F + G}{60} \qquad (2)$$

But in a similar fashion, the code at Reference H, once an hour, will cause the timer to slow down. The delay at Reference C has now become:

$$\text{one second} - B - D - E - \frac{F + G}{60} - \frac{H}{3600} \qquad (3)$$

We have once again succeeded in trading one problem for another. By using the delay of equation (2) or (3) we have, in effect, speeded up the inner loop. This implies that if the timer routine is run for only a short period of time (say, 45 seconds) it will be running fast.

At this point the choices we face are these:

(1) Use equation (1) which is accurate for short runs (less than one minute) but slow for longer runs.

(2) Use equation (2) which is accurate for moderate runs (between one minute and one hour) but fast for shorter runs and slow for longer runs.

(3) Use equation (3) which is accurate for long runs (over one hour) but fast for shorter runs.

Ideally, a general purpose timer routine should be accurate in all timing ranges. Fortunately, this problem also has a solution. The messy solution would be to somehow adjust the delay constant at Reference C depending on whether path P1, P2, or P3 was taken last.

The simple solution is to add two additional delays (one in the P1 path and one in the P2 path) such that all loops take exactly the same length of time to execute.

By incorporating all of the changes described above, the final logic of the timer routine is shown in Figure T.

Having developed a flowchart for the timer routine, the next step is to code it. Each different machine will require its own timer routine; the actual coding of such a routine is left to the reader. However, there are several common considerations that need to be mentioned at this point.

(1) The routine should be written in the machine's native language (i.e., assembly code) and not in BASIC or any other high level language. There are two reasons for this, both relating to the accuracy of the timer. First, it is very difficult to precisely control the speed of a program written in anything but machine language. Second, and more important, is the fact that the speed of a timer subroutine written in, say, BASIC will vary greatly depending on the program it is called from. This is a problem that is intrinsic to interpretive languages like BASIC.

(2) The execution speed of the microcomputer must be constant for the timer to be accurate. Surprisingly enough, most of the personal microcomputers (including the PET 2001 and the TRS-80) do not run at a constant speed because of the way they implement memory refresh. The error this introduces to the timer routine is small but cumulative.

(3) There needs to be an external means of signalling the timer routine to stop running and return the elapsed time (Reference B on the flowchart). The easiest way to do this is for the timer to check continuously for a key being struck on the keyboard. When a key is depressed, the timer routine exits.

(Figure T is on page 12--
the text continues on page 13)

Playing Around

Computers have been doing it for years.

It often seems as if the computer, that number-crunching arbiter of social change, was created expressly for games playing. Its amazing ability to follow the twists and turns of even the most brobdingnagian set of rules has long been prized by accountants, tax consultants--and a certain rare breed of gamesmen and women. Some of the latter have even taken to inventing games that say "not for human usage."

The latest one is the "Time Game," written by staff members of *Popular Computing* magazine in Los Angeles. As is customary with this type of activity, the consumer--in this case, brainy types who shell out upwards of twenty dollars a year for monthly issues that regularly challenge them to tie up their employer's computers solving esoteric problems--is asked to supply all the work to get the game set up.

"Nobody actually plays the game at all," states *Popcom's* editor Fred Gruenberger. "They may read it without playing it, or they may program their computers to play it and not bother to read it, but no one physically plays it in the sense of sitting down to play Monopoly."

Indeed, no one could. A computer is an integral part of the game. (It is possible to construct the "crooked" dice which the game requires, but the permutations of the game itself are so astronomical that only a digital data

processor could play the game out to its conclusion.)

But what about all those people with home computers? Will they be interested in a game that, while it has a playing board, is impossible to follow except as numerical printouts on the computer's viewing screen? "Probably not," says Gruenberger, "but then, the type of person who only plays 'Space Wars' on his home computer is not my audience anyway."

The audience for *Popular Computing* is a loyal band of international computists--Gruenberger's term for people who care more about software and grey matter than about computer hardware--who are scattered across the globe in research centers, universities, embassies, and private sector data processing departments in thirty-one nations. They don't view computer games the same way the average consumer does. "Playing the game isn't really the point," Gruenberger states happily. "It's solving the problem of getting the machine to play the game that counts."

Popcom readers are counting on some other problems for computer games playing which are scheduled for coming issues. These include "The Telephone Game," "The Ph.D Game," and "The Postal Service." All will be so complex that solutions may even be impossible to determine.

This sophisticated view of games and gaming is not without humor. The "Time Game," in its printed-for-humans format, is packed with it. The "Time" of the title refers not to chronology but to the weekly newsmagazine, and the game's playing board is covered with comedic comments about *Time* and its editorial style. Irreverant? Sure. Irrelevant? Definitely. Funny? You bet. And *Time* doesn't mind. When your circulation is as big as ours, it's all a part of the game.

In "Music," a collection of album reviews credits Phil Spector as producer of several Sun Records sessions, claims that the Talking Heads formed with former members of Television, states that Henri Temianka never played in a string quartet, and attributes Respighi's "Church Windows" to Palestrina. (Go back 10 spaces)

In "The Arts," a generally approving review of what is termed "West Coast New Wave Art" contains an illustration that is captioned "Neo-Cubist Canine." On close inspection, it turns out to be Fifi the poodle in the comic strip "Bringing Up Father" by Bill Kavanaugh and Hal Camp. (Go back 20 spaces)

The lead story in "Nation" transposes quotations by Jody Powell and Jimmy Carter, attributes a quote to Jules Germond when he neither wrote nor said any such thing, and through a typographical error makes sense out of a statement made by Edward Kennedy. (Go back 10 spaces) However, all parties proclaim their satisfaction with the piece. (Go forward 3 spaces)

The "Cinema" section profiles "The Ten Best Films of All Time." Three of the selections were viciously panned by "Cinema" when originally released. No mention is made of this fact. (Go back 7 spaces)

In the "Science" section, your lead story ballyhoos a "miracle drug" for the common cold. Although mentioning the drug's unlicensed status and exhorbitant price, the article fails to note that the primary ingredients are aspirin, corn syrup and Darvon. (Go back 20 spaces) No one notices. (Go back 2 more spaces)

Time drops behind *TV Guide* in total advertising revenue. (All players go back one space)

START

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26

Rules

Each player tries to go from START to the cell beyond 50. The distance traveled at each move is given by the toss of two dice, amended by the directions on the game board. The dice are crooked, of course: for each die, the one and six spots come up 2 times out of 8, and spots 2-3-4-5 come up 1 time out of 8, each.

Books," raves are awarded to an elligible novel. The novel doesn't well, but wins the Pulitizer Prize iction. Whereupon you pan both ook and the award. (Go back to RT)

The "Fashion" section's coverage of "the punk look" manages to offend Calvin Klein, *W*, Trashy Lingerie, and Women Against Violence Against Women. (Go back 14 spaces) But circulation improves. (Go forward 9 spaces) A punk fashion cover story is scheduled. (Go back 6 spaces)

"People" (the section, not the magazine) runs Chuck Barris' picture with Rip Taylor's name. (So what. Stay where you are.)

The entire page of the "Religion" section is devoted to an examination of "the electric church." It is discovered that the story is a rehash of a *New West* piece of three months earlier, a *Wall Street Journal* piece of two months earlier, and a *60 Minutes* segment of one month earlier. (Go back 25 spaces) Everyone notices. (Go back 5 more spaces) And everyone laughs. (Go back 5 more spaces)

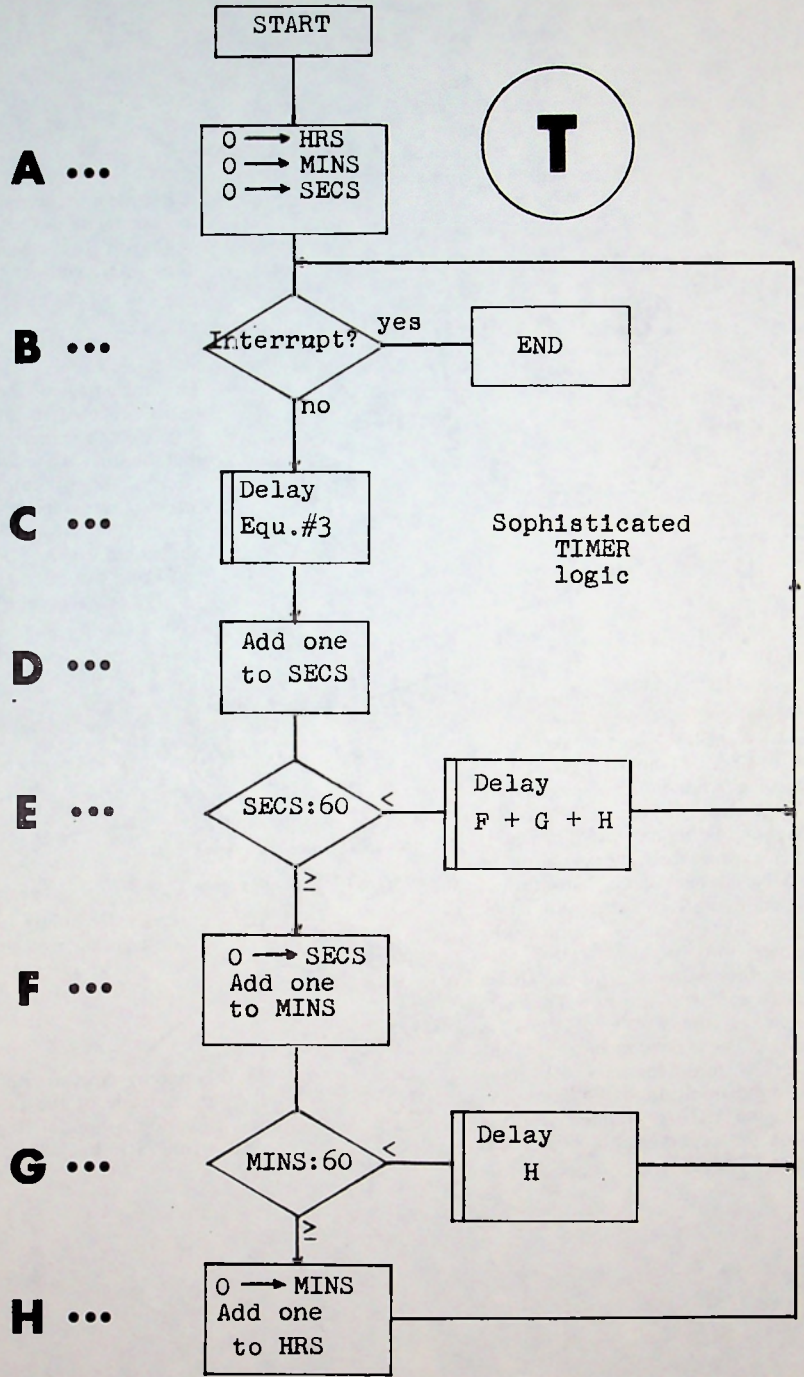
he penny arcade trivia game machines dd this question to their list: "Who as last year's *Time* 'Man of the ear'? (Go back 8 spaces)

The "Psychiatry" section profiles a team of doctors who are developing a "shocking" nine-hundred page report on a nuclear family in which the mother and father met on "The Dating Game," won their honeymoon on "The Newlywed Game," decided to have children after talking things over on "Bedtime Stories," put their daughter into private school with money she won on "Joker Joker Joker," and who are now getting a divorce as the result of "3's A Crowd." The essay concludes that the doctors' final report will have "profound implications." Five days later, nine UCLA pre-med students confess the report was a hoax. (Go back to START)

You insult a 25-year subscriber by repeatedly billing him for his paid-up subscription, and by refusing for months to acknowledge the error. (Go back 9 spaces)

In "Economy--Special Report," no mention is made of the fact that the last "Economy--Special Report" was dead wrong. (Go back 30 spaces)

*


START
A ...
0 → HRS
0 → MINS
0 → SECS
T
B ...
Interrupt?
yes
END
no
C ...
Delay
Equ.#3
Sophisticated
TIMER
logic
D ...
Add one
to SECS
E ...
SECS:60
<
Delay
F + G + H
≥
F ...
0 → SECS
Add one
to MINS
G ...
MINS:60
<
Delay
H
≥
H ...
0 → MINS
Add one
to HRS

(4) In writing the three delay loops, it is very useful to count time in terms of machine cycles. First, divide the processor's clock rate into one second to determine the number of machine cycles in a second. Next, add up the number of machine cycles for each instruction in the computation portion of the timer routine. The difference between these two values is the number of cycles the delay loop must waste. Dividing this number by the total number of cycles consumed by executing the delay loop once gives the loop count. Finally, any left over cycles can be wasted by inserting an appropriate number of NOP's after the delay loop.

(5) Once it has been written, the timer routine needs to be calibrated. This can be done by running the timer alone for at least ten hours and comparing its measurement of the time elapsed with the true value as given by an accurate clock. Any error in the measurement can be corrected by adjusting the delay at flowchart Reference C.

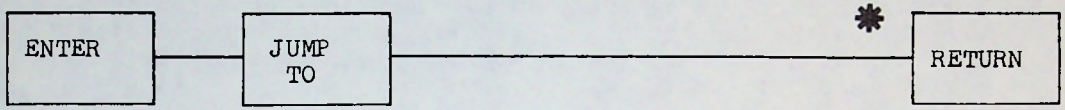
A timer routine for the Apple II computer has been developed according to the above specifications and it has proved to be a very useful tool. It is written in assembly language but is callable from any language. It is accurate to within one second throughout its entire range, due primarily to careful coding and the basic design of the Apple which allows that machine to run at a precise, constant speed.

Similar timer routines could be developed for other personal microcomputers. The accuracy of such routines need only be limited by the technical restrictions of the particular computer.

--Further notes on timing by FG--

Mr. Babcock's clever trick can be used in a slightly different form for a basic timing task; namely, the timing of the operations of whatever computing tool you use. Suppose, for example, you wish to time some of the operation codes of the 6502 microprocessor.

The logic is this: arrange a loop to be traversed exactly one million times. Within that loop is a call to a subroutine, for which the logic is:



This program is run and carefully timed, either by using Babcock's TIMER or by use of an accurate watch. For the latter method, the entire program is executed, say, 20 times, to obtain a reasonably precise time of one execution.

Now, at the place marked (*) in the subroutine, whatever it is that is to be timed can be inserted. For example, to time the 6502's Load Accumulator (LDA) instruction, insert one LDA instruction just before the RETURN, and change the JUMP to point to that LDA. The entire program has thus changed by exactly one instruction. If the program is now again executed 20 times, the execution time will change by the time of 20,000,000 LDA's.

In one such experiment, the original time for 20 executions was 592 seconds, and the time for 20 executions with the LDA inserted was 670 seconds. We can reason then that 20,000,000 LDA's consumed 78 seconds, so that for that op-code, the 6502 runs at around 256,000 operations per second. We must say "around" because of our crude way of timing the 20 executions; an error in reading the clock of one second can have a significant effect on the results.

Flowchart W indicates an easy way to count to a million in 8-bit language, using three words of storage for counters. Counter N on that flowchart counts the individual millions.

Although Babcock warns that timing is tricky in a higher level language (and particularly an interpretive language like most BASICs), it is still of interest to know --even approximately--how fast various BASIC statements execute. The basic logic is given as follows:

```
100  K = 0
110  N = 0
120  G = .987
130  H = .876

200  GØSUB 1000
210  N = N + 1
220  IF N< 500 THEN 200

300  K = K + 1
310  PRINT K
320  GØTØ 110


1000  GØTØ 1100

1100  RETURN
```

If your BASIC can produce an audible beep, do so at line 305; it helps in the timing.

The program as shown is timed. In Applesoft, for example, it runs around 5.125 seconds between beeps. The term "around" has new meaning here, as will be explained later.

Having timed out the basic structure, we can now insert, say,:

```
1090  A = 2 + 3
```

and change

```
1000  GØTØ 1090
```

and time the new program. Suppose that between the printings of each K value the program now takes 6.5 seconds. Then the statement A = 2 + 3, executed 500 times, takes 1.425 seconds, which means that this simple statement executes at the rate of about 350 per second. By changing statement 1090 we can similarly determine that a statement like

```
1090  X = RND(1)
```

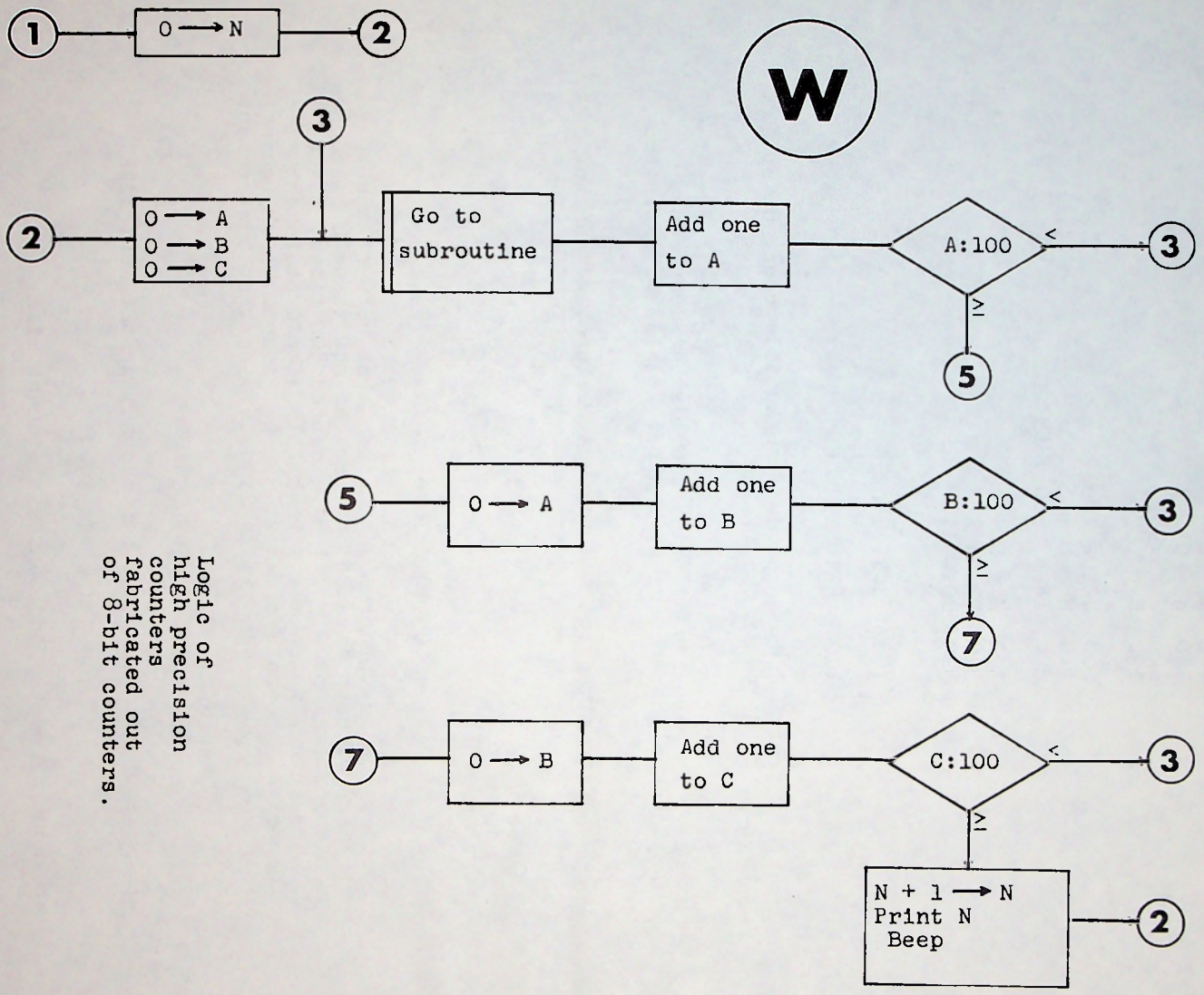
will execute at about 190 per second; a statement like

```
1090  X = ATN(.999)
```

at about 63 per second; and a statement like

```
1090  A = SIN(G) + SQR(H) - CØS(G/H)
```

at about 9.7 per second.



Logic of high precision counters fabricated out of 8-bit counters.

Programs executed in machine language utilize RAM--"Random Access Memory"--within the true meaning of random access; namely, that any word of storage can be accessed in the same time as any other word. But the execution of an interpretive program is entirely different; there are tables to be searched and lists to be refered to, for every statement that is executed.

Consider these two programs in BASIC:

A

```
1000  K = 0
1010  N = 0

1500  GØSUB 2000
1510  N = N + 1
1520  IF N < 1000 THEN 1500
1530  K = K + 1
1540  PRINT K
1550  GØTØ 1010

2000  GØTØ 2500
2500  RETURN
```

B

```
1000  GØTØ 2000
2000  RETURN

4000  K = 0
4010  N = 0

4100  GØSUB 1000
4110  N = N + 1
4120  IF N < 1000 THEN 4100
4130  K = K + 1
4140  PRINT K
4150 GØTØ 4010
```

The two programs purport to do the same thing in much the same way. Yet program A will execute in 11.06 seconds (in Applesoft) and program B in 10.85 seconds, simply because of the placing of the subroutine, and the way a BASIC interpreter goes looking for statements.

KNOCKOUT

That amazing mind of David Ferguson (of the Ferguson Tool Company) has been at it again, busily demolishing elegant computing problems with even more elegant analytic solutions.

This time he has taken on KNOCKOUT (see issue 55 and more recently issue 77, page 19). In this game (devised as a computer game), we start with all the integers from 3 up. The leader of the stream, 3, dictates knocking out the number that is 3 back, namely 6, and replacing it with the 3. Now 4 is the leader, which dictates knocking out the number that is 4 back, namely 8, and replacing it with the 4.

This process continues indefinitely. What number will be knocked out at each stage, K?

Mr. Ferguson's solution is as follows:

Define f(x) as the highest power of 2 that divides x.

Let n be the number knocked out at the Kth step.

If $K \equiv 1 \pmod 3$ then

n = K + 5 if $K \equiv 4 \pmod 9$ or if f(K+5) is odd;

otherwise n = (K+5)/3.

If $K \not\equiv 1 \pmod 3$ then

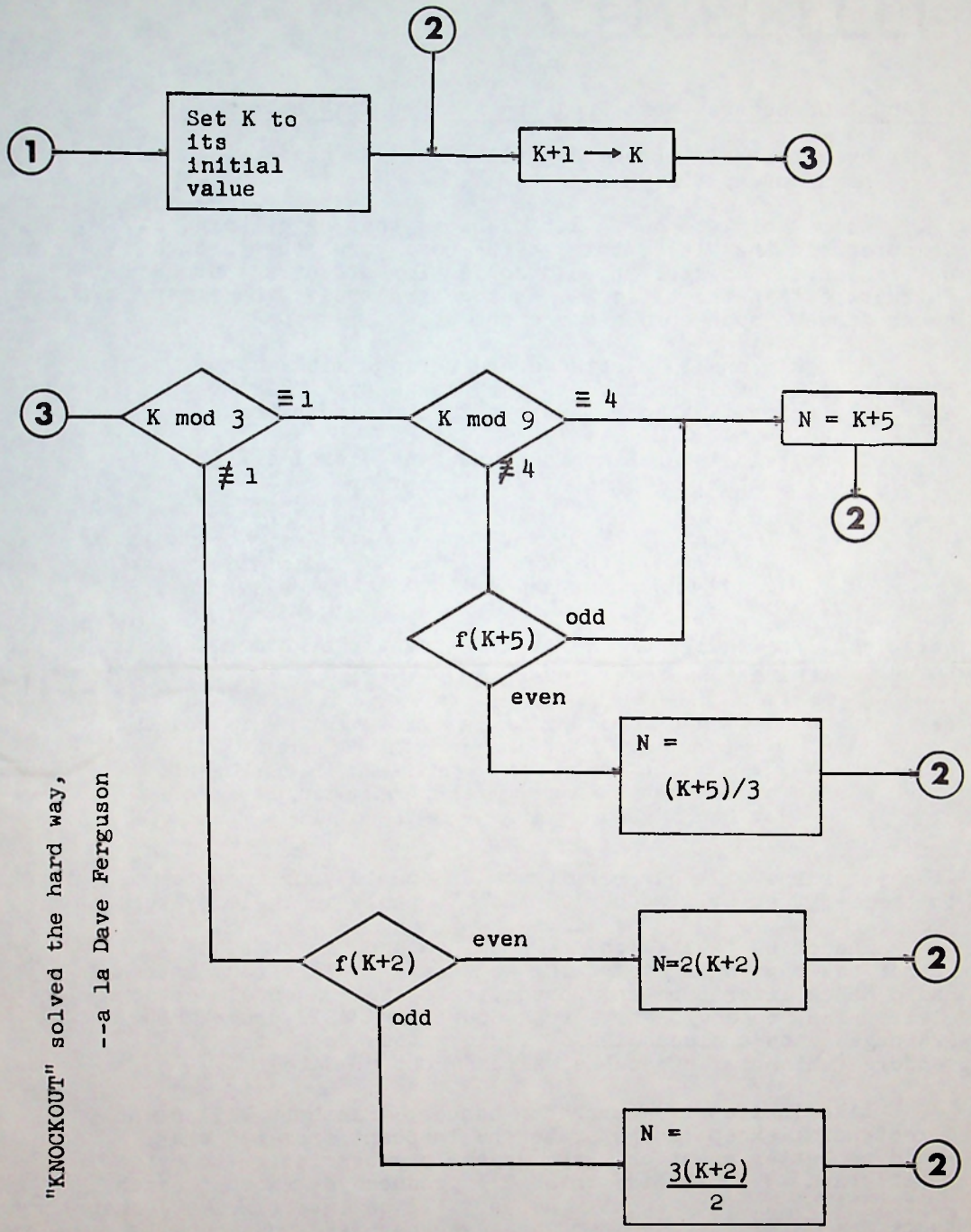
n = 2(K + 2) if f(K+2) is even;

otherwise n = 3(K+2)/2.

His solution is also shown in flowchart form. It lends itself to integer BASIC, given the MOD function. If your BASIC lacks the MOD function, one of the following may do to simulate the operation A mod B:

M = A - (A/B)*B

M = A - B*INT(A/B)



"KNOCKOUT" solved the hard way,
--a la Dave Ferguson

BOOK REVIEW

Personal Computers--What They Are and How To Use Them

by Byron Wels, Prentice-Hall 1978, 193 pages, hard cover, $12.95.

We're going to see a lot of books that have "Personal Computer" in the title; this is the same virus that brought out, a year or so ago, any old collection of words that had "structured" in the title. As a general rule, the first books of each such epidemic are the worst.

But this one is a strange and curious mish-mash. First of all, it is printed in very large type, like Dick and Jane, which is a poor way to bulk out a thin book.

Then, Wels intermixes fact and fancy carelessly, as if he didn't know the difference. For example:

> The Apple II computer is a household appliance that will water your lawn, control your budget, and turn your heat or air-conditioning on and off.

which could be true, but the implication for a novice is that the Apple comes that way. Or this:

> Fred works for an airline, and as a result, gets reduced fares for himself and his family. He carefully plans his vacations around his computer--and the airline's! By dialing up the correct telephone number, he can couple his family computer directly to the airline's computer.

If any airline will now permit you to couple your computer to theirs, they will soon plug that loophole in their system.

The chief fault with the book, though, is that the author has thrown in everything he knows. This ranges from Radio Shack literature and photos, to a section on elementary electronics, to a glossary, to a course in BASIC programming (24 pages), to a discussion of logic gates, to a list of vendors (which is, of course, already out of date).

Like ineffective cures for cancer, this book will do no direct harm, except to separate the innocent from $13 that would be better spent on, say, a year's subscription to Byte or Kilobaud. Books like this will produce yet another batch of bright-eyed high school kids who will believe (in 1980) that there are talking typewriters because they read in a book that such things existed.